

Правительство Российской Федерации

**Федеральное государственное автономное образовательное
учреждение**

высшего профессионального образования

“Национальный исследовательский университет

“Высшая школа экономики”

Московский институт электроники и математики Национального
исследовательского университета "Высшая школа экономики"

Отчёт по лабораторной работе

по дисциплине «Программирование алгоритмов защиты информации»

Выполнил:

Студент группы СКБ202

Мотыгуллин Амир Булатович

ОГЛАВЛЕНИЕ

<i>Задача:</i>	3
<i>Код программы</i>	3
<i>Тест</i>	5

ЗАДАЧА:
ПДСЧ на основе блочного шифра и все встроить libakrypt

Код ПРОГРАММЫ:

ak_random

```
/** @brief Класс для хранения данных о шифре и текущем значении счетчика */
typedef struct magma_random_data {
    /** @brief Используемый шифр */
    ak_bckey magma;
    /** @brief Текущее внутреннее состояние генератора */
    ak_uint64 current;
    /** @brief Последние 8 сгенерированный байт */
    ak_uint8 last_bytes[8];
    /** @brief Количество выданных байт из последних 8 сгенерированных */
    ak_uint8 current_index;
} magma_rnd;
```

``typedef struct magma_random_data { ... } magma_rnd;``:

Это объявление структуры. ``typedef`` создает новый тип данных ``magma_rnd``, который является синонимом ``struct magma_random_data``. Это улучшает читаемость кода, делая его более лаконичным.

``ak_bckey magma;``:

Это член структуры, представляющий собой экземпляр структуры ``ak_bckey``. Содержит информацию о ключе шифра Magma, необходимую для его работы.

``ak_uint64 current;``:

Это 64-битное целое без знака (``ak_uint64``), представляющее текущее внутреннее состояние ГПСЧ. Это значение используется как входные данные для шифра Magma при генерации псевдослучайных чисел. Изменение этого значения при каждом вызове генерации обеспечивает последовательность различных выходных данных.

``ak_uint8 last_bytes[8];``:

Это массив из 8 байтов без знака (``ak_uint8``). В этот массив записываются 8 байтов псевдослучайных данных, сгенерированных шифром Magma за одну итерацию. Генератор выдает байты из этого массива по одному, пока он не будет пуст.

``ak_uint8 current_index;``:

Это байт без знака (``ak_uint8``), представляющий индекс текущего байта в массиве ``last_bytes``, который будет возвращен при следующем запросе случайного числа. Когда ``current_index`` достигает 8, это означает, что все 8 байтов из ``last_bytes`` были использованы, и необходимо сгенерировать новую порцию случайных данных.

```
static int ak_random_magma_randomize_ptr(ak_random rnd, const ak_pointer ptr, const ssize_t size)
{
    magma_rnd* ctx = NULL;
    if (rnd == NULL) return ak_error_message(ak_error_null_pointer, __func__,
        "use a null pointer to a random generator");
    if (ptr == NULL) return ak_error_message(ak_error_null_pointer, __func__,
        "use a null pointer to initial vector");
    if (size != 8) return ak_error_message(ak_error_wrong_length, __func__,
        "use initial vector with wrong length");

    /* текущая реализация ограничена 32-мя битами, поэтому больше 4х октетов не копируем */
    ctx = rnd->data.ctx;
    ctx->current = *(ak_uint64*)ptr;
    ctx->current_index = 8;

    return ak_error_ok;
}
```

`static int ak_random_magma_randomize_ptr(...)`:

Объявление статической функции. `static` означает, что функция видна только внутри текущего файла. Функция возвращает целое число, которое является кодом ошибки.

`magma_rnd* ctx = NULL;`:

Объявление указателя `ctx` на структуру `magma_rnd`. Эта структура (определенная ранее) хранит внутреннее состояние ГПСЧ. Указатель инициализируется значением `NULL`.

`if (rnd == NULL) ...`:

Проверка на `NULL` указатель на контекст генератора (`rnd`). Если указатель `NULL`, значит, генератор не инициализирован, и функция возвращает код ошибки, используя вспомогательную функцию `ak_error_message`.

`if (ptr == NULL) ...`:

Проверка на `NULL` указатель на начальный вектор (`ptr`). Если указатель `NULL`, значит, начальный вектор не предоставлен, и функция возвращает код ошибки.

`if (size != 8) ...`:

Проверка размера начального вектора (`size`). Если размер не равен 8 байтам, возвращается код ошибки. Начальный вектор должен быть 64-битным числом (8 байт = 64 бита).

`ctx = rnd->data.ctx;`:

Получение указателя на структуру `magma_rnd` из контекста генератора (`rnd`).

`ctx->current = *(ak_uint64*)ptr;`:

Инициализация внутреннего состояния генератора (`ctx->current`). Значение начального вектора (`ptr`) интерпретируется как 64-битное беззнаковое целое число (`ak_uint64`) и присваивается переменной `ctx->current`. Это ключевая строка, которая устанавливает seed для ГПСЧ.

`ctx->current_index = 8;`:

Установка `current_index` в 8. Это переменная, отслеживающая количество уже использованных байтов из буфера `last_bytes` в структуре `magma_rnd`.

`return ak_error_ok;`:

Возвращение кода успешного выполнения.

```
static int ak_random_magma_next(ak_random rnd)
{
    magma_rnd* ctx = (magma_rnd*)rnd->data.ctx;

    ak_bkey_encrypt_ecb(ctx->magma, &ctx->current, ctx->last_bytes, 8);
    ctx->current++;
    ctx->current_index = 0;

    return ak_error_ok;
}
```

Функция `ak_random_magma_next`, которая генерирует следующий блок псевдослучайных байтов в генераторе случайных чисел, основанном на шифре Magma:

1. `static int ak_random_magma_next(ak_random rnd)`:

`static` означает, что функция видна только внутри текущего файла. Функция возвращает целое число (`int`), которое, указывает на успех (`ak_error_ok`) или ошибку.

2. `magma_rnd* ctx = (magma_rnd*)rnd->data.ctx;`:

Эта строка получает указатель на структуру `magma_rnd`. Она делает это, используя указатель `rnd` на структуру `ak_random`, а затем `rnd->data.ctx` для доступа к члену `data.ctx` структуры `ak_random`. `data.ctx` содержит указатель на структуру `magma_rnd`, которая хранит данные, специфичные для генератора на основе Magma. Результат приводится к типу `magma_rnd*` с помощью приведения типов.

3. `ak_bkey_encrypt_ecb(ctx->magma, &ctx->current, ctx->last_bytes, 8);`:

Это ядро функции. Вызывается функция `ak_bkey_encrypt_ecb` для шифрования. Разберем аргументы:

`ctx->magma`:

Указатель на структуру `ak_bkey`, содержащую ключ и параметры шифрования Magma. Получается из структуры `ctx`.

`&ctx->current`:

Указатель на 64-битное целое число (`ak_uint64`) `ctx->current`. Это значение используется как входные данные для шифрования. Важно отметить, что *этот* `current` является внутренним состоянием генератора. После каждого вызова функции оно изменяется.

`ctx->last_bytes`:

Указатель на массив `ctx->last_bytes` размером 8 байтов. В этот массив записываются зашифрованные данные — результат работы `ak_bkey_encrypt_ecb`. Это и есть сгенерированный блок псевдослучайных чисел.

`8`:

Размер блока данных (в байтах), который нужно зашифровать.

4. `ctx->current++;`:

Увеличивает значение `ctx->current` на 1. Это изменяет внутреннее состояние генератора, гарантируя, что следующий вызов `ak_random_magma_next` сгенерирует другой набор псевдослучайных чисел.

5. `ctx->current_index = 0;`:

Эта строка обнуляет `ctx->current_index`. Этот член структуры используется для отслеживания того, сколько байтов из `ctx->last_bytes` уже использовано. Обнуление указывает на то, что все 8 байтов в `ctx->last_bytes` готовы к использованию.

6. `return ak_error_ok;`:

Функция возвращает значение `ak_error_ok`, сигнализируя об успешном выполнении.

```

static int ak_random_magma_random(ak_random rnd, const ak_pointer buffer, ssize_t size)
{
    if (rnd == NULL) return ak_error_message(ak_error_null_pointer, __func__,
        "use a null pointer to a random generator");
    if (buffer == NULL) return ak_error_message(ak_error_null_pointer, __func__,
        "use a null pointer to data");
    if (size <= 0) return ak_error_message(ak_error_wrong_length, __func__,
        "use a data vector with wrong length");

    ak_uint8* tempBuffer = (ak_uint8*)buffer;
    ssize_t addedCount = 0;
    magma_rnd* ctx = (magma_rnd*)rnd->data.ctx;

    while (size > 0)
    {
        addedCount = 8;
        if (addedCount > size)
            addedCount = size;
        if (addedCount > 8 - ctx->current_index)
            addedCount = 8 - ctx->current_index;

        if (addedCount > 0)
            memcpy(tempBuffer, ctx->last_bytes + ctx->current_index, addedCount);

        tempBuffer += addedCount;
        ctx->current_index += addedCount;
        size -= addedCount;

        if (ctx->current_index >= 8)
            rnd->next(rnd);
    }

    return ak_error_ok;
}

```

Функция `ak_random_magma_random`, которая заполняет предоставленный буфер случайными байтами, используя генератор случайных чисел (ГСЧ), основанный на шифре Magma:

1. **static int ak_random_magma_random(ak_random rnd, const ak_pointer buffer, ssize_t size):**
static означает, что функция видна только внутри текущего файла. Функция возвращает целое число (`int`), которое указывает на результат выполнения: `ak_error_ok` — успех, другие значения — ошибки. Аргументы:
 rnd:
 Указатель на структуру `ak_random`, которая содержит контекст ГСЧ (внутреннее состояние).
 buffer:
 Указатель на буфер, который нужно заполнить случайными данными.
 ak_pointer — скорее всего, `void*` (универсальный указатель).
 size:
 Размер буфера в байтах.
2. **Проверка на ошибки:**
 Следующие три `if` проверяют корректность входных данных:
 if (rnd == NULL) ...:
 Проверяет, что указатель на контекст ГСЧ не нулевой.
 if (buffer == NULL) ...:
 Проверяет, что указатель на буфер не нулевой.
 if (size <= 0) ...:
 Проверяет, что размер буфера больше нуля. При ошибке вызывается функция `ak_error_message` и функция возвращает код ошибки.
3. **ak_uint8* tempBuffer = (ak_uint8*)buffer;:**
 Указатель `buffer` (тип `void*`) приводится к типу `ak_uint8*`, что позволяет работать с ним как с массивом байтов. `tempBuffer` указывает на начало буфера.
4. **ssize_t addedCount = 0;:**

Объявляется переменная `addedCount`, которая хранит количество байтов, скопированных в буфер за одну итерацию цикла.

5. **`magma_rnd* ctx = (magma_rnd*)rnd->data.ctx;`**

Извлекается указатель на структуру `magma_rnd`, которая содержит внутреннее состояние ГСЧ, основанного на Magma.

6. **`while (size > 0):`**

Цикл `while` повторяется до тех пор, пока весь буфер не будет заполнен (`size` станет 0).

7. **`addedCount = 8; ...:`**

Внутри цикла определяется, сколько байтов можно скопировать в буфер на текущей итерации:

`addedCount = 8;`

Начальное значение — 8 байтов.

`if (addedCount > size) addedCount = size;`

Если оставшийся размер буфера меньше 8 байтов, `addedCount` уменьшается до оставшегося размера.

`if (addedCount > 8 - ctx->current_index) addedCount = 8 - ctx->current_index;`

Если во внутреннем буфере ГСЧ осталось меньше 8 байтов, `addedCount` уменьшается до количества доступных байтов.

8. **`if (addedCount > 0) memcpy(tempBuffer, ctx->last_bytes + ctx->current_index, addedCount);`**

Если `addedCount` больше нуля, `memcpy` копирует `addedCount` байтов из внутреннего буфера ГСЧ (`ctx->last_bytes`, начиная с индекса `ctx->current_index`) в буфер пользователя (`tempBuffer`).

9. **Обновление указателей и счетчиков:**

`tempBuffer += addedCount;`

Указатель `tempBuffer` сдвигается на `addedCount` байтов вперед.

`ctx->current_index += addedCount;`

Индекс текущего байта во внутреннем буфере ГСЧ увеличивается.

`size -= addedCount;`

Оставшийся размер буфера уменьшается.

10. **`if (ctx->current_index >= 8) rnd->next(rnd);`**

Если во внутреннем буфере ГСЧ закончились байты (`ctx->current_index` достигло 8), вызывается функция `rnd->next(rnd)` для генерации нового блока случайных байтов. Эта функция обновляет внутреннее состояние ГСЧ.

11. **`return ak_error_ok;`**

Функция возвращает `ak_error_ok`, сигнализируя об успешном завершении.

```
static int ak_random_magma_free(ak_random rnd)
{
    if (rnd == NULL) return ak_error_message(ak_error_null_pointer, __func__,
        "use a null pointer to a random generator");
    if (rnd->data.ctx != NULL)
        free(rnd->data.ctx);

    return ak_error_ok;
}
```

Функция `ak_random_magma_free`, которая освобождает память, выделенную для контекста генератора случайных чисел, основанного на алгоритме Magma.

1. `static int ak_random_magma_free(ak_random rnd)`:

`static` означает, что функция видна только внутри текущего файла. Функция принимает один аргумент:

`rnd`:

Указатель на структуру `ak_random`, которая содержит контекст генератора случайных чисел.

2. `if (rnd == NULL) return ak_error_message(ak_error_null_pointer, func, "use a null pointer to a random generator");`:

Это проверка на NULL. Если указатель `rnd` равен NULL (т.е., указывает на пустую память), функция выводит сообщение об ошибке с помощью `ak_error_message` и возвращает код ошибки, сигнализируя о том, что передан невалидный указатель.

3. `if (rnd->data.ctx != NULL)`:

Проверяет, существует ли контекст генератора. `rnd->data.ctx` — это поле структуры `ak_random`, которое хранит указатель на контекст `magma_rnd`, содержащий данные, специфичные для генератора Magma. Проверка нужна для того, чтобы избежать попытки освободить NULL-указатель, что может привести к сбою программы.

4. `free(rnd->data.ctx);`:

Если контекст (`rnd->data.ctx`) не равен NULL, эта строка освобождает память, на которую он указывает, используя стандартную функцию `free`. Это возвращает память операционной системе.

5. `return ak_error_ok;`:

Функция возвращает `ak_error_ok`, сигнализируя об успешном выполнении.


```

int ak_random_create_magma(ak_random generator, ak_bckey magma)
{
    int error = ak_error_ok;

    if ((error = ak_random_create(generator)) != ak_error_ok)
        return ak_error_message(error, __func__, "wrong initialization of random generator");

    generator->next = ak_random_magma_next;
    generator->randomize_ptr = ak_random_magma_randomize_ptr;
    generator->random = ak_random_magma_random;
    generator->free = ak_random_magma_free;

    generator->data.ctx = (struct magma_random_data*)malloc(sizeof(magma_rnd));
    ((magma_rnd*)generator->data.ctx)->magma = magma;
    ak_uint64 value = ak_random_value();
    ak_random_magma_randomize_ptr(generator, &value, 8);
    return error;
}

```

Функция `ak_random_create_magma`, которая инициализирует генератор случайных чисел (ГСЧ) на основе шифра Magma.

1. **int ak_random_create_magma(ak_random generator, ak_bckey magma):**

Функция возвращает целое число (int), которое представляет код ошибки или `ak_error_ok` в случае успеха. Аргументы:

`generator`: Указатель на структуру `ak_random`, которая будет представлять инициализированный ГСЧ.

`magma`: Указатель на контекст ключа шифра Magma (`ak_bckey`). Этот ключ будет использоваться генератором для создания псевдослучайных чисел.

2. **int error = ak_error_ok;**

Инициализируется переменная `error` значением `ak_error_ok`, указывающим на отсутствие ошибок на данный момент.

3. **if ((error = ak_random_create(generator)) != ak_error_ok):**

Вызывается функция `ak_random_create(generator)`. Функция выполняет базовую инициализацию структуры `ak_random`, выделяет необходимую память. Если `ak_random_create` возвращает код ошибки (не `ak_error_ok`), то это значение записывается в `error`, и выполнение функции `ak_random_create_magma` прерывается.

4. **return ak_error_message(error, func, "wrong initialization of random generator");:**

Если произошла ошибка на предыдущем шаге, выводится сообщение об ошибке с помощью `ak_error_message` и возвращается код ошибки.

5. **generator->next = ak_random_magma_next;**

Устанавливаются указатели на функции, необходимые для работы ГСЧ. `ak_random_magma_next` — это функция, которая генерирует следующий блок случайных байтов.

6. **generator->randomize_ptr = ak_random_magma_randomize_ptr;**

`ak_random_magma_randomize_ptr` — функция, которая инициализирует генератор, используя указатель на данные.

7. **generator->random = ak_random_magma_random;**

`ak_random_magma_random` — функция, которая заполняет буфер случайными байтами.

8. **generator->free = ak_random_magma_free;**

`ak_random_magma_free` — функция, которая освобождает память, используемую ГСЧ.

9. **generator->data.ctx = (struct magma_random_data*)malloc(sizeof(magma_rnd));:**

Выделяется память для структуры `magma_rnd`, которая хранит внутреннее состояние ГСЧ Magma (например, ключ, счетчик и т.д.).

10. **((magma_rnd*)generator->data.ctx)->magma = magma;**

В выделенную память записывается указатель на контекст ключа Magma (`magma`).

11. **ak_uint64 value = ak_random_value();:**

Генерируется случайное 64-битное значение с помощью функции `ak_random_value()`.

12. **ak_random_magma_randomize_ptr(generator, &value, 8);:**

Вызывается функция `ak_random_magma_randomize_ptr` для инициализации ГСЧ, используя сгенерированное 64-битное значение. 8 — размер инициализирующего блока в байтах.

13. **return error;**

Возвращается значение `error`. Если все прошло успешно, то будет возвращено

ak_error_ok.

```
738
739  /*! \brief Инициализация контекста генератора на основе Магмы */
740  dll_export int ak_random_create_magma(ak_random, ak_bckey);
741
742  /*
```

Объявление функции `ak_random_create_magma` на языке C.:

`dll_export`:

Он указывает компилятору, что данная функция должна быть экспортирована из DLL (Dynamic Link Library) — динамически подключаемой библиотеки. Это позволяет другим программам использовать эту функцию, загружая DLL. Если этот макрос отсутствует, функция будет доступна только внутри текущего модуля.

`int`:

Это тип возвращаемого значения функции. Функция возвращает целое число, которое, вероятно, служит кодом возврата: 0 (или `ak_error_ok` в контексте libakgrpt) означает успех, а другие значения указывают на разные виды ошибок.

`ak_random_create_magma`:

Это имя функции. Она создает и инициализирует генератор случайных чисел (ГСЧ) на основе криптографического алгоритма Magma.

```

1 #include <stdio.h> //Подключение стандартной библиотеки ввода/вывода.
2 #include <libakrypt.h> //Подключение заголовочного файла библиотеки libakrypt.
3 #include <unistd.h> //Подключение библиотеки для работы с POSIX-системой
4
5 int main(void)
6 {
7     int error = ak_error_ok; //Инициализация переменной для хранения кодов ошибок.
8     int exitstatus = EXIT_FAILURE; //Инициализация статуса выхода программы значением, указывающим на неудачу.
9     struct bkey ctx; //Объявление структуры `bkey` для хранения контекста ключа блочного шифрования.
10    struct random rnd; //Объявление структуры `random` для хранения контекста генератора случайных чисел.
11
12    ak_uint8 generated[800]; //Объявление массива для хранения 800 сгенерированных байт.
13    ak_uint8 key[32] = { 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07, 0x08,0x09,0x0A,0x0B, 0x0C,0x0D,0x0E,0x0F,
14                        0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17, 0x18,0x19,0x1A,0x1B, 0x1C,0x1D,0x1E,0x1F, };
15    //Объявление и инициализация массива `key` 32-байтовым ключом.
16    if (ak_libakrypt_create(NULL) != ak_true) {
17        ak_libakrypt_destroy();
18        return EXIT_FAILURE;
19    }
20    //Инициализация библиотеки libakrypt. Если инициализация не удалась, программа завершается с кодом ошибки.
21
22    ak_bkey_create_magma(&ctx); //Создание контекста ключа на основе Магмы.
23    ak_bkey_set_key(&ctx, key, 32); //Установка ключа в контекст `ctx`.
24    ak_random_create_magma(&rnd, &ctx); //Создание генератора псевдослучайных чисел на основе Магмы, используя контекст ключа
    `ctx`.
25    rnd.random(&rnd, generated, 800); //Генерация 800 псевдослучайных байт и сохранение их в массив `generated`.
26    printf("%s\n\n", ak_ptr_to_hexstr(generated, 40, ak_false));
27    //Вывод первых 40 байт сгенерированных данных в шестнадцатеричном формате.
28    printf("%s\n\n", ak_ptr_to_hexstr((ak_uint8*)generated + 760, 40, ak_false));
29    //Вывод последних 40 байт сгенерированных данных (с 760 по 799) в шестнадцатеричном формате.
30    rnd.free(&rnd); //Освобождение ресурсов, занятых генератором случайных чисел.
31
32    if (error == ak_error_ok) exitstatus = EXIT_SUCCESS;
33    //Если не было ошибок, статус выхода программы устанавливается в `EXIT_SUCCESS`
34    ak_libakrypt_destroy(); //Освобождение ресурсов, занятых библиотекой libakrypt.
35    return exitstatus;
36 }
37

```

1. **int main(void):**
Это функция `main`, точка входа в программу.
2. **int error = ak_error_ok;:**
Объявляется переменная `error`, которая хранит код ошибки. Инициализируется значением `ak_error_ok`, обозначающим отсутствие ошибок.
3. **int exitstatus = EXIT_FAILURE;:**
Объявляется переменная `exitstatus`, которая хранит код возврата программы. Инициализируется значением `EXIT_FAILURE` (1), обозначающим неудачное завершение.
4. **struct bkey ctx;:**
Объявляется переменная `ctx` типа `struct bkey`. Это структура представляет контекст ключа шифрования для Магма.
5. **struct random rnd;:**
Объявляется переменная `rnd` типа `struct random`. Это структура, представляющая контекст ГСЧ.
6. **ak_uint8 generated[800];:**
Объявляется массив `generated` размером 800 байт типа `ak_uint8` (беззнаковый 8-битный целый тип). Этот массив будет хранить сгенерированные случайные данные.
7. **ak_uint8 key[32] = { ... };:**
Объявляется и инициализируется массив `key` размером 32 байта, содержащий 32-байтовый ключ для шифра Магма.
8. **if (ak_libakrypt_create(NULL) != ak_true):**
Вызывается функция `ak_libakrypt_create(NULL)`. Функция инициализации библиотеки `libakrypt`. Если функция возвращает значение, отличное от `ak_true` (ложь), то происходит ошибка инициализации.
9. **ak_libakrypt_destroy(); return EXIT_FAILURE;:**
Если инициализация библиотеки неудачна, библиотека освобождается с помощью `ak_libakrypt_destroy()`, и программа завершается с кодом ошибки `EXIT_FAILURE`.
10. **ak_bkey_create_magma(&ctx);:**
Создается контекст ключа Магма (`ctx`) с помощью функции `ak_bkey_create_magma`.

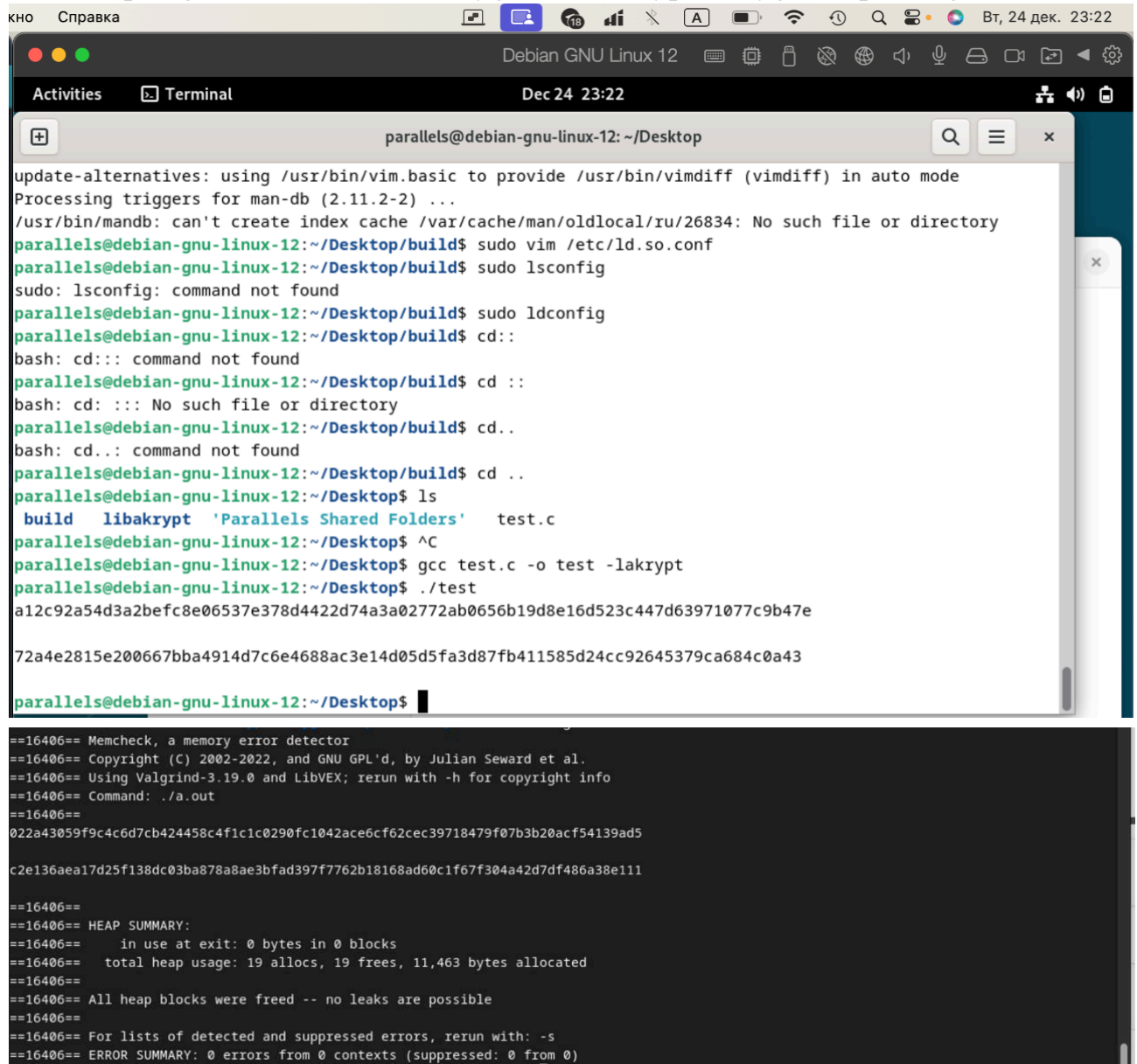
11. `ak_bckey_set_key(&ctx, key, 32);`
Устанавливается ключ для контекста Magma (`ctx`) с помощью функции `ak_bckey_set_key`.
12. `ak_random_create_magma(&rnd, &ctx);`
Создается и инициализируется ГСЧ Magma (`rnd`) с использованием созданного контекста ключа (`ctx`).
13. `rnd.random(&rnd, generated, 800);`
Генерируются 800 байтов случайных данных и записываются в массив `generated` с помощью функции `rnd.random`.
14. `printf("%s\n\n", ak_ptr_to_hexstr(generated, 40, ak_false));`
Выводятся первые 40 байтов сгенерированных данных в шестнадцатеричном формате с помощью функции `ak_ptr_to_hexstr`.
15. `printf("%s\n\n", ak_ptr_to_hexstr((ak_uint8*)generated + 760, 40, ak_false));`
Выводятся последние 40 байтов сгенерированных данных в шестнадцатеричном формате.
16. `rnd.free(&rnd);`
Освобождается память, занятая ГСЧ, с помощью функции `rnd.free`.
17. `if (error == ak_error_ok) exitstatus = EXIT_SUCCESS;`
Если переменная `error` равна `ak_error_ok`, то `exitstatus` устанавливается в `EXIT_SUCCESS` (обычно 0), обозначающее успешное завершение.
18. `ak_libakrypt_destroy();`
Освобождаются ресурсы библиотеки `libakrypt`.
19. `return exitstatus;`
Возвращается код завершения программы.

```

4  #include <stdio.h>
5  #include <libakrypt.h>
6  #include <unistd.h>
7
8  int main(void)
9  {
10     /* , */
11     int error = ak_error_ok;
12     /* */
13     int exitstatus = EXIT_FAILURE;
14     /* */
15     struct bckey ctx;
16     struct random rnd;
17     /* */
18
19     ak_uint8 generated[800];
20     ak_uint8 key[32] = { 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07, 0x08,0x09,0x0A,0x0B, 0x0C,0x0D,0x0E,0x0F,
21                        0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17, 0x18,0x19,0x1A,0x1B, 0x1C,0x1D,0x1E,0x1F, };
22
23     /* */
24     if (ak_libakrypt_create(NULL) != ak_true) {
25         ak_libakrypt_destroy();
26         return EXIT_FAILURE;
27     }
28
29     ak_bckey_create_magma(&ctx);
30     ak_bckey_set_key(&ctx, key, 32);
31     ak_random_create_magma(&rnd, &ctx);
32     rnd.random(&rnd, generated, 800);
33     printf("%s\n\n", ak_ptr_to_hexstr(generated, 40, ak_false));
34     printf("%s\n\n", ak_ptr_to_hexstr((ak_uint8*)generated + 760, 40, ak_false));
35     rnd.free(&rnd);
36
37     /* */
38     if (error == ak_error_ok) exitstatus = EXIT_SUCCESS;
39     ak_bckey_destroy(&ctx);
40     ak_libakrypt_destroy();
41     return exitstatus;
42 }
```

Нужно не забыть, очень важно добавить `ak_bkey_destroy(&ctx)` для освобождения ресурсов, связанных с ключом `ctx` после того, как он больше не нужен. Это предотвратит утечки памяти.

<https://git.miem.hse.ru/abmotygullin/libakrypt-motygullin-pcdch.git>



```
кно Справка Debian GNU Linux 12 Вт, 24 дек. 23:22
Activities Terminal Dec 24 23:22
parallels@debian-gnu-linux-12: ~/Desktop
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vimdiff (vimdiff) in auto mode
Processing triggers for man-db (2.11.2-2) ...
/usr/bin/mandb: can't create index cache /var/cache/man/oldlocal/ru/26834: No such file or directory
parallels@debian-gnu-linux-12:~/Desktop/build$ sudo vim /etc/ld.so.conf
parallels@debian-gnu-linux-12:~/Desktop/build$ sudo lsconfig
sudo: lsconfig: command not found
parallels@debian-gnu-linux-12:~/Desktop/build$ sudo ldconfig
parallels@debian-gnu-linux-12:~/Desktop/build$ cd::
bash: cd::: command not found
parallels@debian-gnu-linux-12:~/Desktop/build$ cd ::
bash: cd: ::: No such file or directory
parallels@debian-gnu-linux-12:~/Desktop/build$ cd..
bash: cd.: command not found
parallels@debian-gnu-linux-12:~/Desktop/build$ cd ..
parallels@debian-gnu-linux-12:~/Desktop$ ls
build  libakrypt  'Parallels Shared Folders'  test.c
parallels@debian-gnu-linux-12:~/Desktop$ ^C
parallels@debian-gnu-linux-12:~/Desktop$ gcc test.c -o test -lakrypt
parallels@debian-gnu-linux-12:~/Desktop$ ./test
a12c92a54d3a2befc8e06537e378d4422d74a3a02772ab0656b19d8e16d523c447d63971077c9b47e
72a4e2815e200667bba4914d7c6e4688ac3e14d05d5fa3d87fb411585d24cc92645379ca684c0a43

parallels@debian-gnu-linux-12:~/Desktop$

==16406== Memcheck, a memory error detector
==16406== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==16406== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==16406== Command: ./a.out
==16406==
022a43059f9c4c6d7cb424458c4f1c10290fc1042ace6cf62cec39718479f07b3b20acf54139ad5
c2e136aea17d25f138dc03ba878a8ae3bfad397f7762b18168ad60c1f67f304a42d7df486a38e111

==16406==
==16406== HEAP SUMMARY:
==16406==    in use at exit: 0 bytes in 0 blocks
==16406==   total heap usage: 19 allocs, 19 frees, 11,463 bytes allocated
==16406==
==16406== All heap blocks were freed -- no leaks are possible
==16406==
==16406== For lists of detected and suppressed errors, rerun with: -s
==16406== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Отлично, все работает правильно работает, утечек нет.